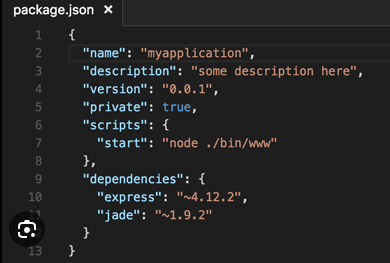
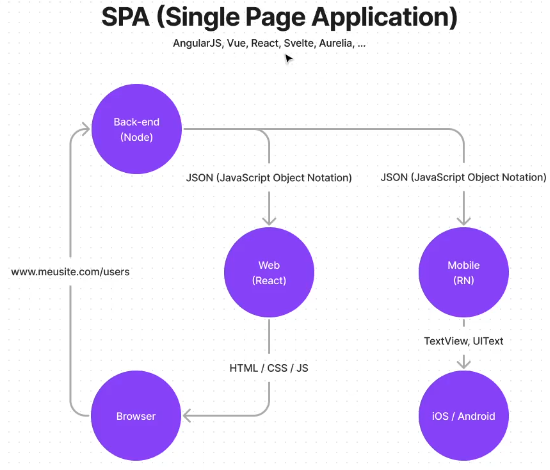
Entender Json: Json e um arquivo universal, que todas as linguagens conseguem utiliza-lo por sua simplicidade, basicamente é uma estrutura simples de arrays, objetos e atributos.

Observe: 

  
Bundlers e compilers

Babel

Compilers: compila JS atual para um JS antigo casa seja necessário por conta de um navegador não suportar a versão.

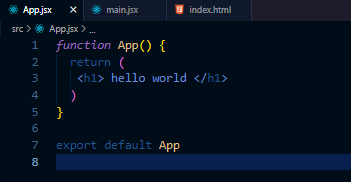
Bundlers:   
Vipe importa módulos na aplicação  
npx degit user/project#main simple-calculator

cd my-project

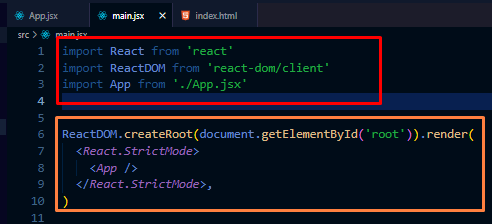
npm install

npm run dev

**Fundamentos react**



**Observe que a funcao apenas retorna o titulo em html, veja que tem JS e HTML juntos, por conta do .jsx extensao do js.**



Importa o react, ele quem vai controlar os módulos.  
 Importa o reactDOM ((Document Object Model) o controlador que pega os objetos HTML e os representa em JS (queryselector, elmentbyid etc)

Cria e renderiza o elemento principal no html com base no elemento raiz da pagina html, seria o que contem o id root

**Ele renderiza o App, gerando o HTML criado em App.js**

**------------COMPONENTES------------**

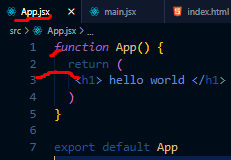
**Muito importante!**

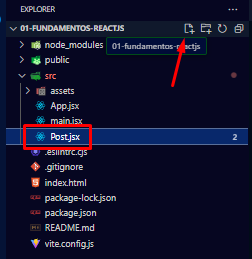
Componentes são utilizados em todo o projeto React.

A ideia e ter um Código em uma parte da aplicação reutiliza-lo/repeti-los sempre que necessário na aplicação inclusive com informação diferentes e que ele se torne adaptável a cada uso, também fica mais fácil de dar manutenção, por ser apenas um pedaço da aplicação.

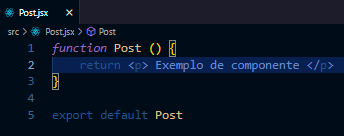
Componente é um arquivo que abstrai parte da interface em outro arquivo e isso pode ser repetido diversas vezes durante o app.

**Identificando componente no React:**

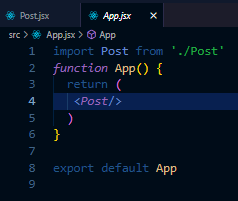
Basicamente uma função que retorna HTML. (JSX = JS + XML):  


Criando, exportando, importando e usando um componente:  


Crie um novo arquivo com a extensão .jsx, exemplo o Post .jsx a cima. **Os componentes sempre devem começar com LETRA MAIÚSCULA**

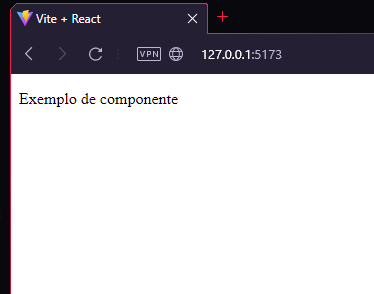


Crie a função com o nome do componente e retorne o HTML desejado, no caso o <p>. Depois no final do código, exporte com o nome da função



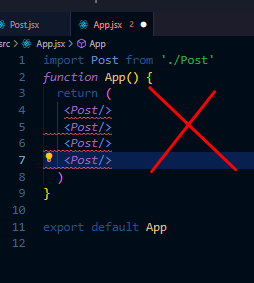
No seu app/pagina onde quer utilizar o componente importe o componente Post (leve em consideração o caminho)

Depois adicione o componente como se fosse uma TAG XML.

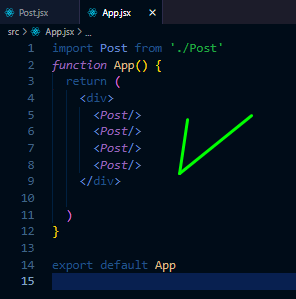


Resultado a cima.

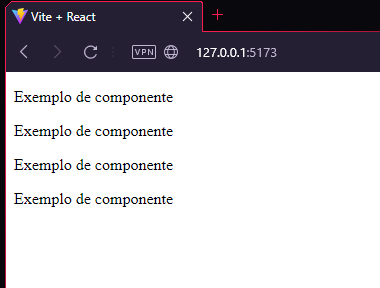
Podemos chamar o componente Post no App quantas vezes quisermos, mas se for utilizar mais de um componente, e necessário ter um elemento pai de referência, ou seja, apenas inserir uma div;



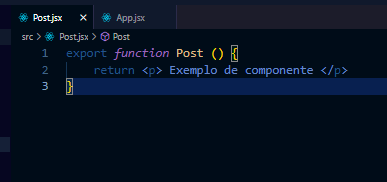
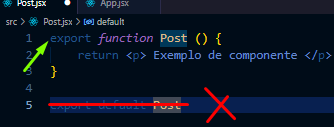
O exemplo a cima não funciona e o baixo funciona.



Resultado:

  
**IMPORTANTE: O método de importação a cima pode acarretar erros, pois o nome do import pode ser alterado e ignorar o nome do export, fazendo com que você possa confundir os components.**

**Seguir método a baixo:**



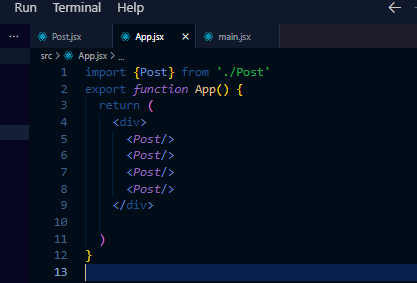
Apenas insira um export antes da function e exclua a exportação padrão.  
Para importar basta inserir um import:

(aperte CTRL + ESPAÇO irá mostrar todos os components disponíveis para utilizar)



(Sempre considere o diretório ./CAMINHO’)

Resultado final:

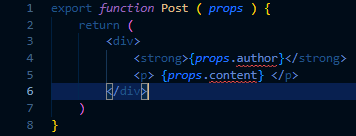


caso você esqueça de alterar o nome do componente na hora de importar, a importação não irá funcionar, diferente do default export, que importaria do mesmo jeito.

**Propriedades dos components:**

Propriedades são informações/conteúdo que eu posso passar aos componentes. Estas propriedades podem ter um visual semelhante o que justifica criar o componente, mas pode conter um conteúdo diferente.

Como no HTML a tag <img src=’exemplo.png’ > tem o search que mostra o caminho da imagem, no mesmo princípio que pode ter a mesma imagem várias vezes ou imagens diferentes as propriedades dos componentes também funcionam da mesma forma.

Na pratica:

|  |
| --- |
| No React o parâmetro **props** (props de propriedade, poderia ser outro nome, mas geralmente e utilizado props pela comunidade React, clean code, boas práticas etc.)  Com isso o componente passa a ser um objeto. |
| Agora podemos deixar cada tag editável, nomeando uma propriedade única para cada parte do componente. Basta chamar o objeto props e adicionar um ‘.nomeDaProriedade’ |
| props.author virou uma propriedade, podendo ser alterada toda vez que se chamar o componente.  E o props.content também. |

No React não se importa CSS no HTML e sim no JSX

Escopo de CSS atrelados ao componente, para estilizar somente aquele componente, sem afetar outros componentes, CSS ESCOPADO: [CSS MODULES](https://github.com/css-modules/css-modules)

Não é necessário baixar pois Vit já contém CSS modules.

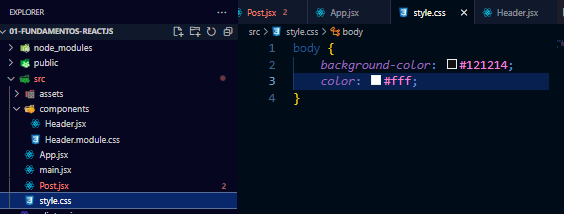
Cria o arquivo com a extensão module e CSS ex: style.module.css

Quando utilizar .module sempre chame os marcadores por classe .nomedaClasse, sempre que possível.

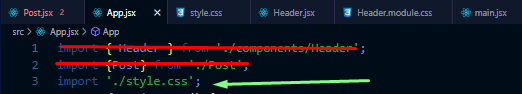
No COMPONENTE.JSX Utilizar ClassName em vez de class do HTML pois o React pode confundir o ‘class’ que também existe no JS.  
Sempre que utilizar nome.modules.css e necessário nomear o import do style, diferente se usar apenas o nome.css que basta apenas inserir o caminho.

CSS Modules criam classes com nomes únicos usando hashes, evitando que estilos de um componente afetem outros. Funciona de forma parecida das propriedades do React.

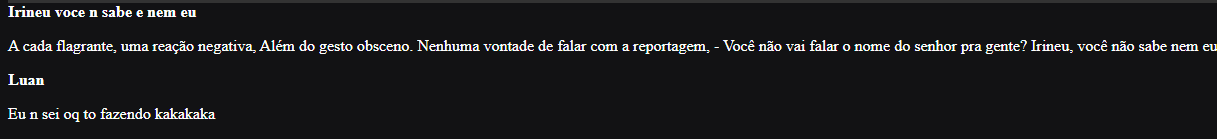
Passo a passo, estilização simples no CSS:

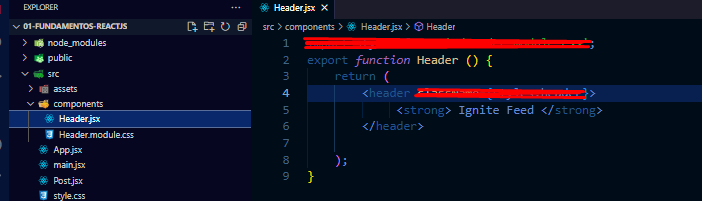
Primeiro crie um arquivo style.css em SRC, mesmo nível no arquivo principal App.jsx e faça uma simples estilização.  


Importamos ele diretamente no App.jsx e não no HTML.

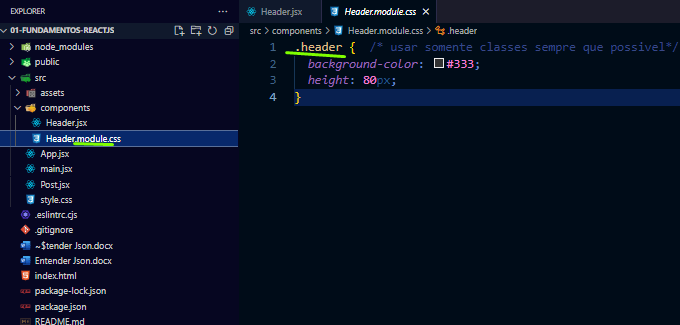


Observe que é apenas necessário listar o caminho.  
Com isso já obtemos o resultado desejado:



Porém esse método irá estilizar o App.jsx como um todo. O React + Vit nos permite editar cada componente independentemente sempre que quisermos e reutilizar esses componentes com estilos diferentes, parecido as propriedades que vimos anteriormente.  
utilizando module.css, veja:  
Primeiro criamos um novo componente.  


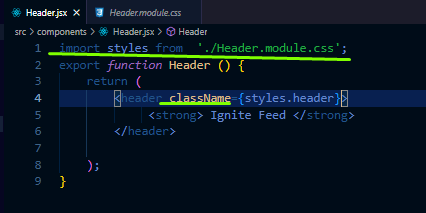
Agora criamos o arquivo style, mas desta vez, adicionamos a extensão .module.css em vez de apenas .css:

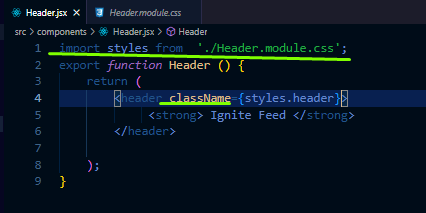
E inserimos a estilização desejada.  


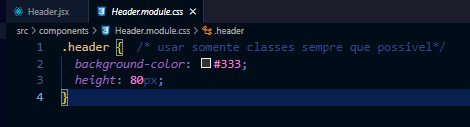
Observe que chamamos pelo identificador de classe .header  
Sempre que possível iremos chamar pela classe por boas práticas.

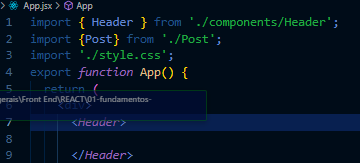
De volta ao componente iremos importar o css, mas desta vez o import precisa de um nome.

E ao inserir o identificador de ‘class’ iremos colocar de ‘className’ para o React não confundir com class de JS.

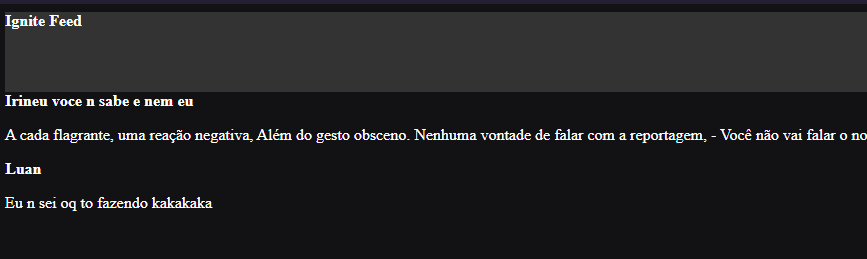


Observe que importamos o style igual um componente e também o utilizamos igual uma propriedade, usando o nomeDaImportação.nomeDaClasseCss.  
observe:

Parte inferior do formulário

Agora basta chamar o componente criado no App.jsx, e ver o resultado:  


Resultado:

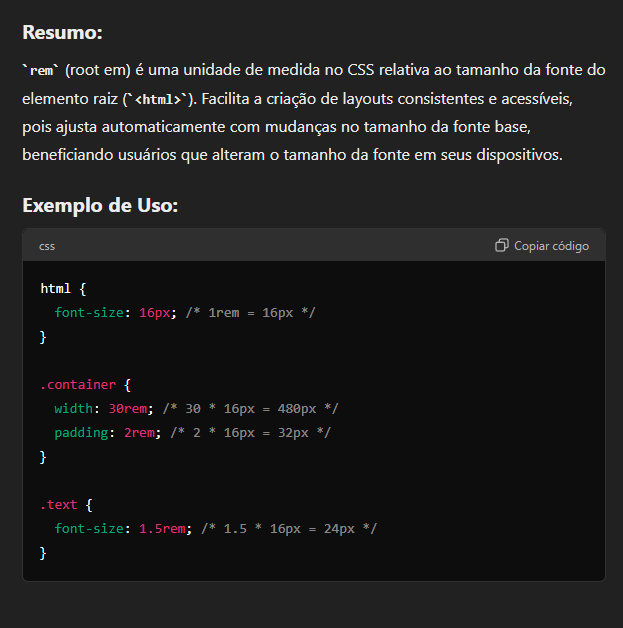


Utilizar variáveis para todas as cores.

Se tiver mais de um tom, faça uma numeração de 100 a 1000 para definir o tom do mais claro ao mais escuro ex: --gray-100 --gray-300 etc.

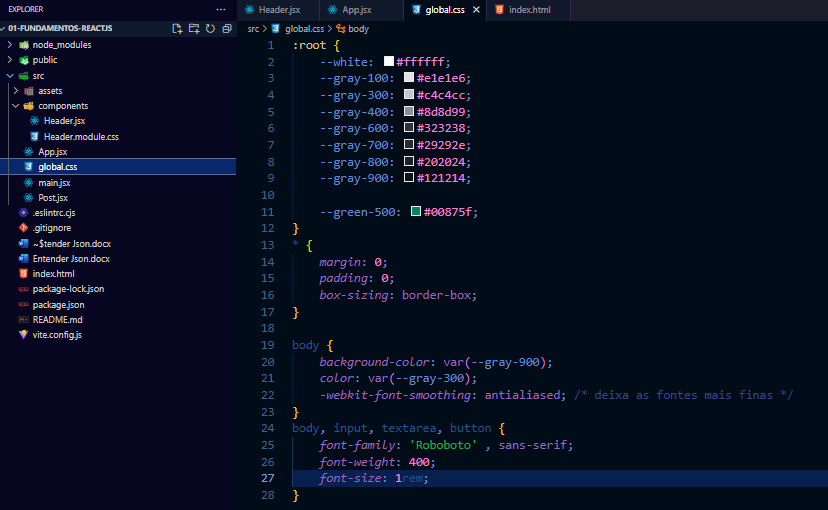
Use os links do google diretamente no HTML, mas deixe eles em primeiro, para garantir que eles carreguem antes da aplicação.

Body, input, textarea, button: Padronizar o tamanho unidades de medidas relativas ex: 1rem, por acessibilidade, por exemplo pessoas aumentam a fonte do celular, nosso app também vai acompanhar.

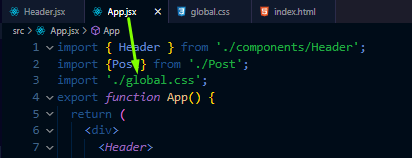


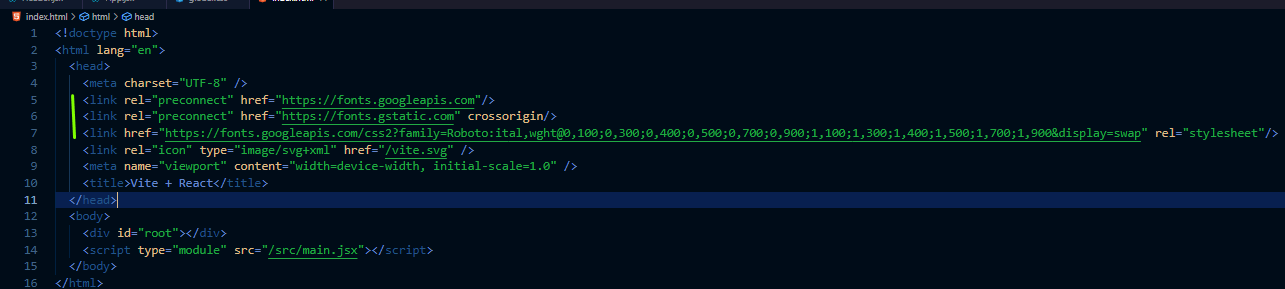
Algumas coisas na prática fica um padrão, por isso iremos renomear o arquivo style.css criado antes para global.css

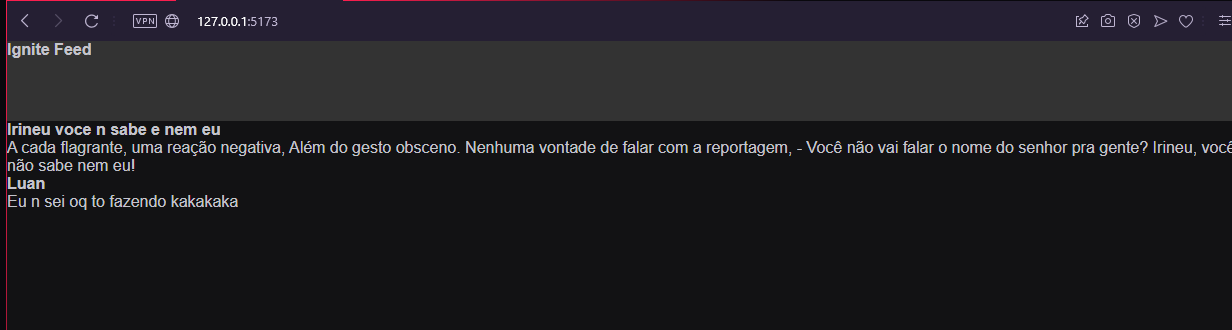
E inserir variáveis de estilo padrão e alguns padrões de inputs, textos, botão etc:



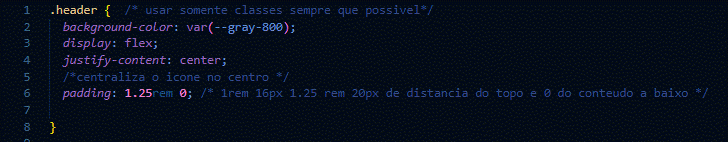
Na importação do App.jsx também alteraremos o nome.



Iremos também pegar a fonte no google fonts roboto e importar no HTML antes de todos os links para evitar erros:  


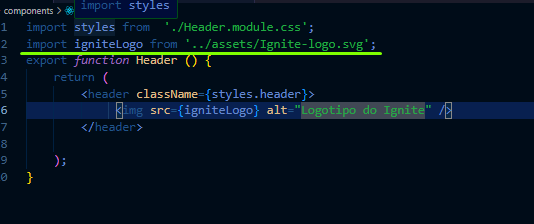
Resultado:  


Agora iremos construir esse layout do figma: [Layout do curso.](https://www.figma.com/design/3wo7UATnhuqTyZcdxSGSkF/Ignite-Feed-(Community)?node-id=1-35&t=r2y9HH0M5lp0XgMp-0)  
Começando pelo básico o logo do cabeçalho. (header).

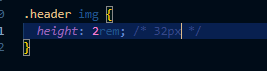
Em Header.module.css  


Esse trecho centraliza o cabeçalho e deixa o conteúdo distanciado do topo.

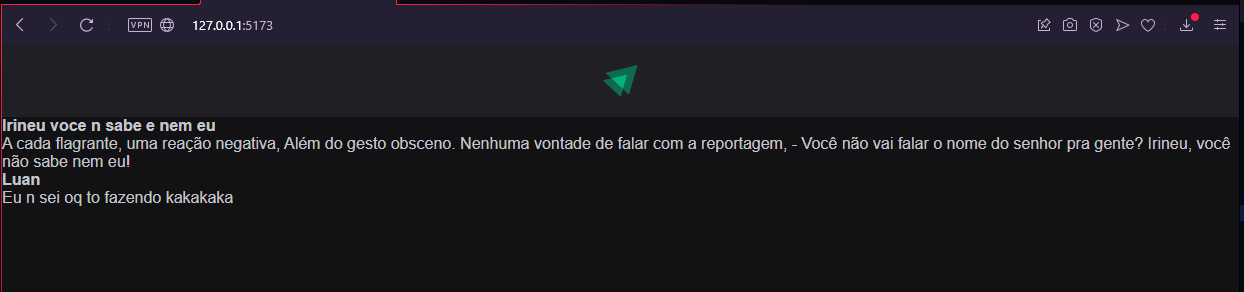
exportamos o logo do figma no formado svg e salvamos em assets e renomeamos para ‘Ignite-logo’

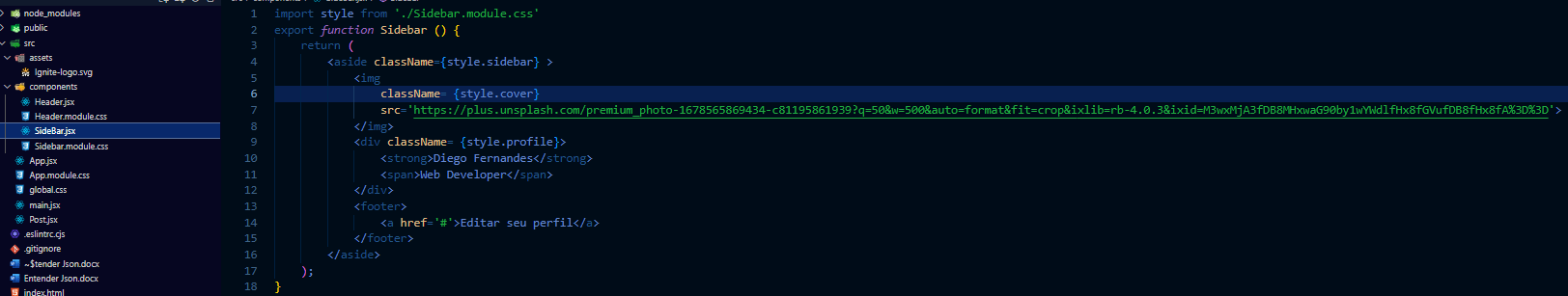
Para usarmos a imagem, importamos ela igual um estilo ou um componente:  


E Chamamos ela como uma variável como nos componentes e estilos.

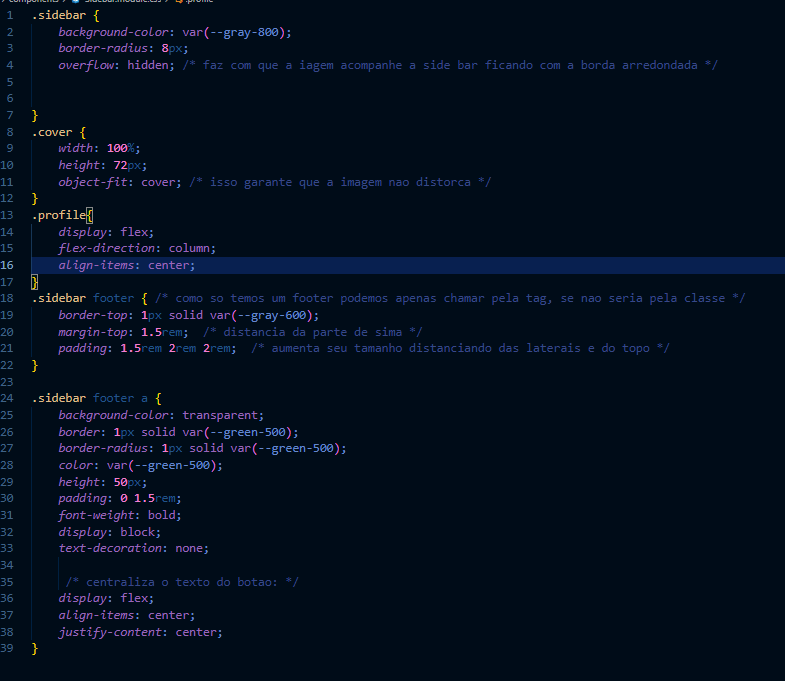
Vamos diminuir um pouco o tamanho do logo:  


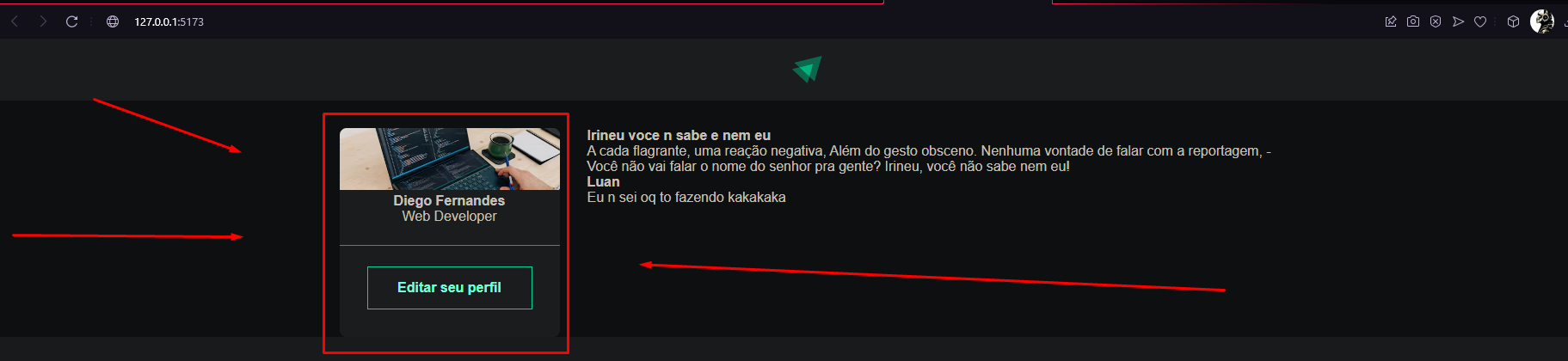
Resultado:



Continuando criando o componente SideBar (um espaço onde fica o perfil).  
Crie o SideBar.jsx e Sidebar.css  
em SideBar.jsx:  


A imagem uma <img> seria a imagem que o usuário colocaria de fundo imagem retirada de bancos de dados gratuitos, depois apenas um título que vai conter o nome e o cargo.



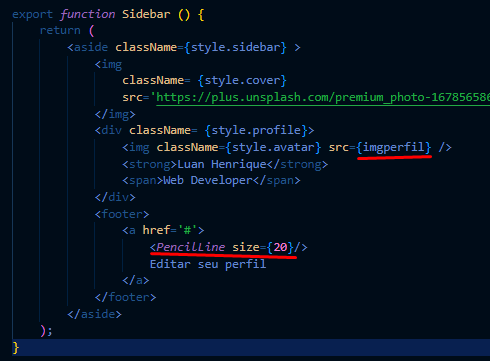
O css a cima separa a side bar e sempre utilizo unidade de medidas rem  
resultado destacado em vermelho:  


Continuando irei inserir uma imagem de perfil e um ícone.

No caso usei o [phosphor-icons](https://github.com/phosphor-icons/homepage), onde apenas e necessário instala-lo no diretório raiz do projeto com npm i phosphor-react e depois importamos o ícone exato que queremos, em vez de importar uma pagina inteira de ícones, só importamos o que iremos usar.



Penciline e o nome do ícone, chamamos ele como um componente em nosso código e a imagem como variável:



Alteramos alguns detalhes para o tamanho da imagem atravessar a div de sima, colocando uma margem negativa, com um cálculo e uma animação de transição

